Advanced JUnit Testing Exercises

Exercise 1: Parameterized Tests

Scenario:

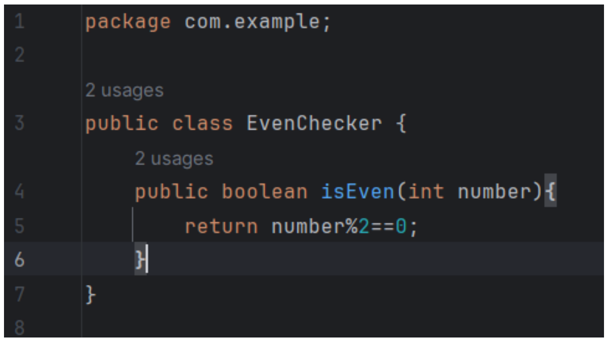
You want to test a method that checks if a number is even. Instead of writing multiple test cases, you will use parameterized tests to run the same test with different inputs.

Steps:

1. Create a new Java class `EvenChecker` with a method `isEven(int number)`.

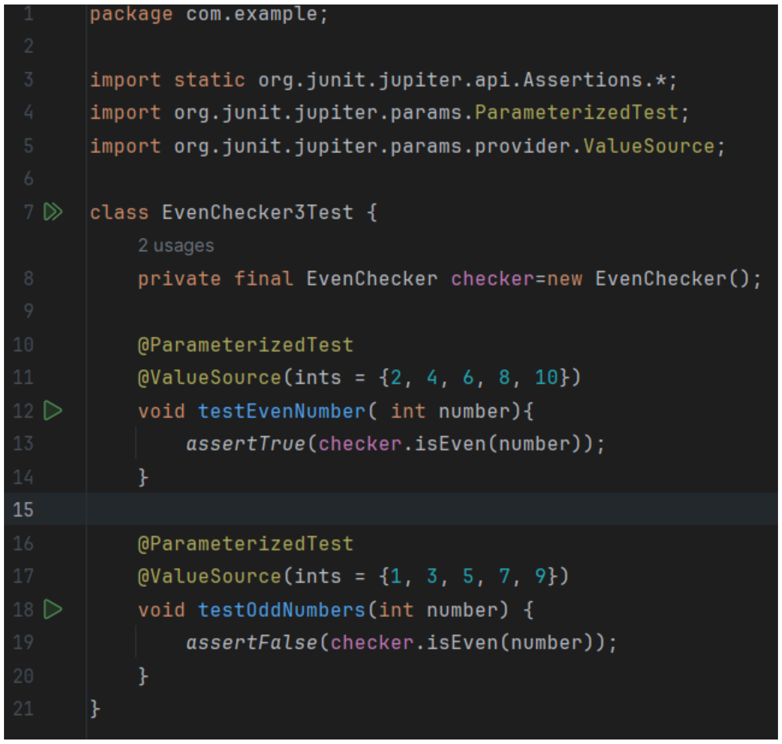
2. Write a parameterized test class `EvenCheckerTest` that tests the `isEven` method with different inputs.

3. Use JUnit's `@ParameterizedTest` and `@ValueSource` annotations.



```java
package com.example;

2 usages
public class EvenChecker {
    2 usages
    public boolean isEven(int number){
        return number%2==0;
    }
}
```



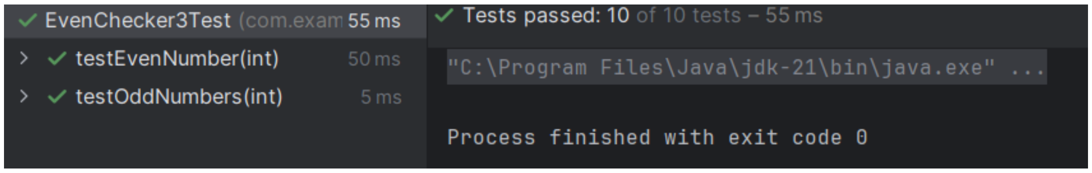
```java
package com.example;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

class EvenChecker3Test {
    2 usages
    private final EvenChecker checker=new EvenChecker();

    @ParameterizedTest
    @ValueSource(ints = {2, 4, 6, 8, 10})
    void testEvenNumber( int number){
        assertTrue(checker.isEven(number));
    }

    @ParameterizedTest
    @ValueSource(ints = {1, 3, 5, 7, 9})
    void testOddNumbers(int number) {
        assertFalse(checker.isEven(number));
    }
}
```



```
✓ EvenChecker3Test (com.exam 55 ms      ✓ Tests passed: 10 of 10 tests – 55 ms
  > ✓ testEvenNumber(int)     50 ms     "C:\Program Files\Java\jdk-21\bin\java.exe" ...
  > ✓ testOddNumbers(int)      5 ms
                                        Process finished with exit code 0
```

Exercise 2: Test Suites and Categories

Scenario:

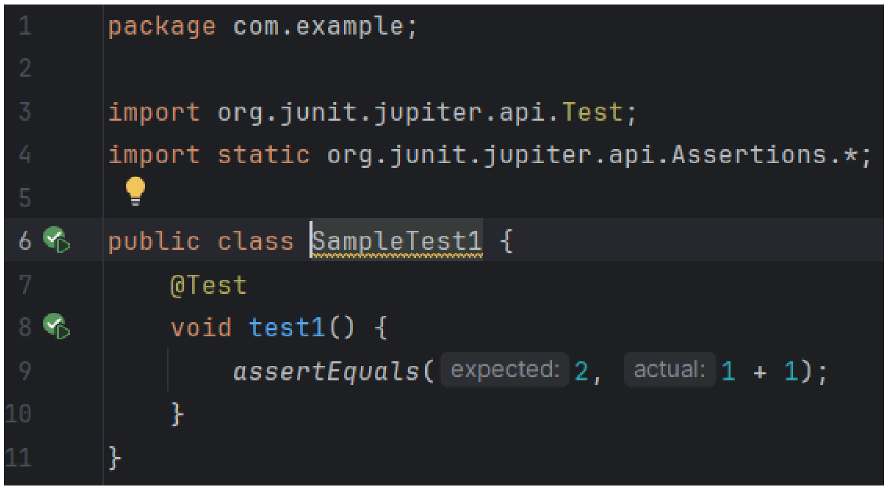
You want to group related tests into a test suite and categorize them.

Steps:

1. Create a new test suite class `AllTests`.

2. Add multiple test classes to the suite.

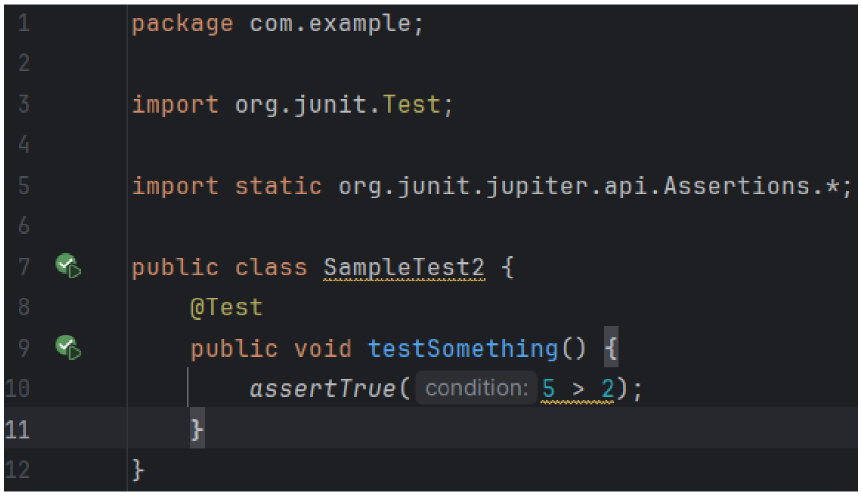
3. Use JUnit's `@Suite` and `@SelectClasses` annotations.



```java
package com.example;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class SampleTest1 {
    @Test
    void test1() {
        assertEquals( expected: 2,  actual: 1 + 1);
    }
}
```

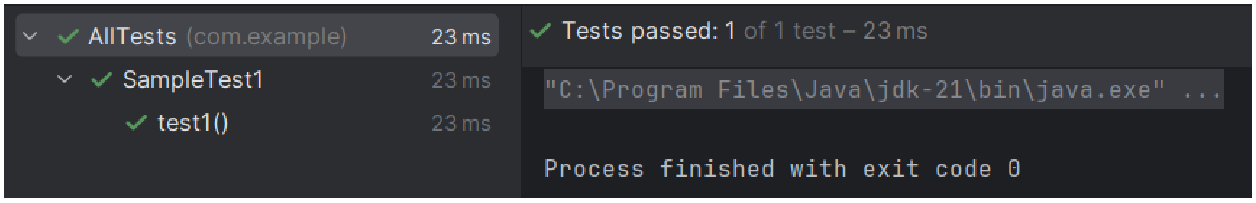


```java
package com.example;

import org.junit.Test;

import static org.junit.jupiter.api.Assertions.*;

public class SampleTest2 {
    @Test
    public void testSomething() {
        assertTrue( condition: 5 > 2);
    }
}
```



```
✓ AllTests (com.example)   23 ms
  ✓ SampleTest1            23 ms
    ✓ test1()              23 ms
```

```
✓ Tests passed: 1 of 1 test – 23 ms
"C:\Program Files\Java\jdk-21\bin\java.exe" ...

Process finished with exit code 0
```

Exercise 3: Test Execution Order

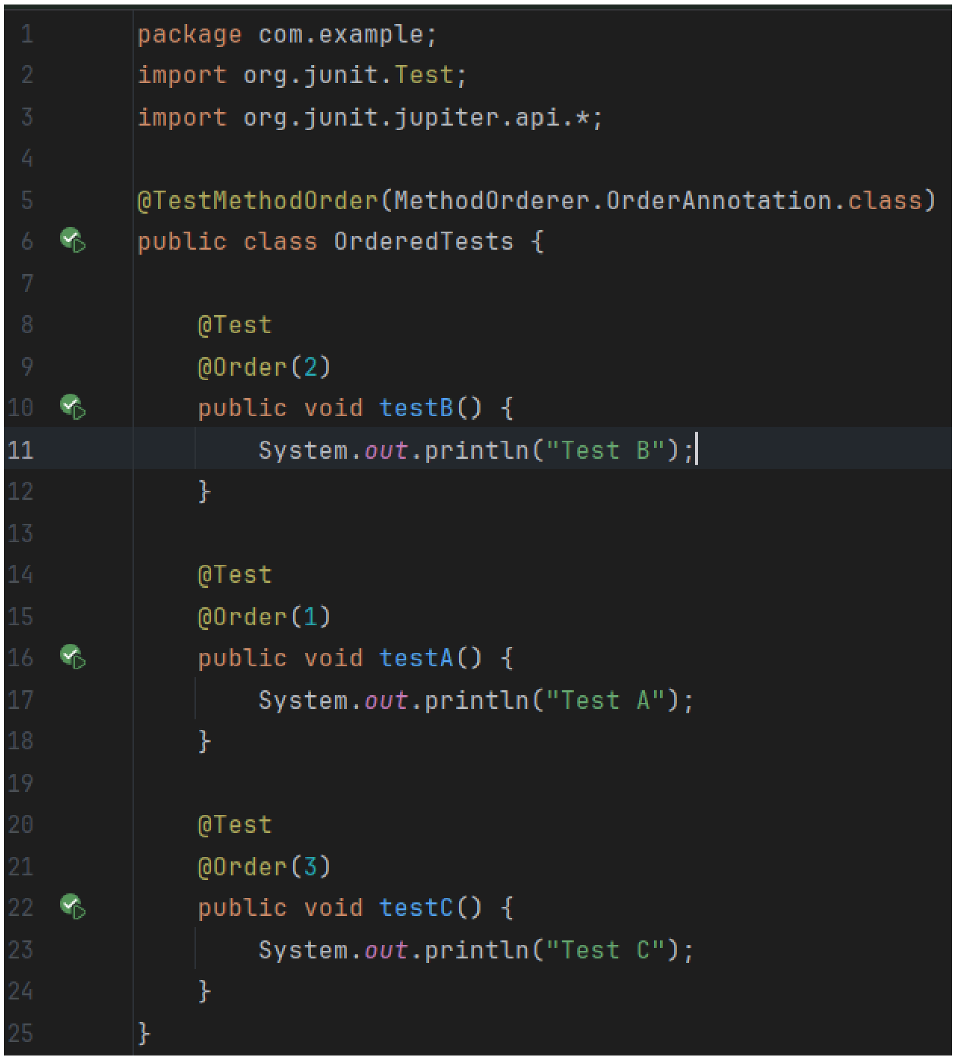
Scenario:

You want to control the order in which tests are executed.

Steps:

1. Create a test class `OrderedTests`.

2. Use JUnit's `@TestMethodOrder` and `@Order` annotations.



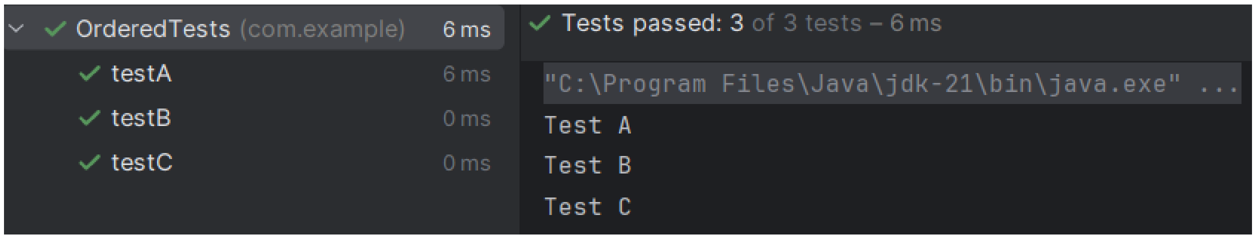
```java
package com.example;
import org.junit.Test;
import org.junit.jupiter.api.*;

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class OrderedTests {

    @Test
    @Order(2)
    public void testB() {
        System.out.println("Test B");
    }

    @Test
    @Order(1)
    public void testA() {
        System.out.println("Test A");
    }

    @Test
    @Order(3)
    public void testC() {
        System.out.println("Test C");
    }
}
```



```
OrderedTests (com.example)  6 ms      Tests passed: 3 of 3 tests – 6 ms
  testA                     6 ms      "C:\Program Files\Java\jdk-21\bin\java.exe" ...
  testB                     0 ms      Test A
  testC                     0 ms      Test B
                                      Test C
```

Exercise 4: Exception Testing

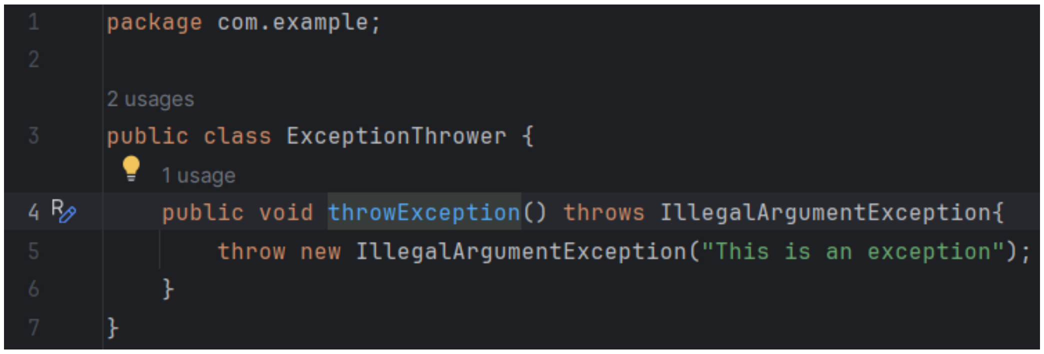
Scenario:

You want to test that a method throws the expected exception.

Steps:

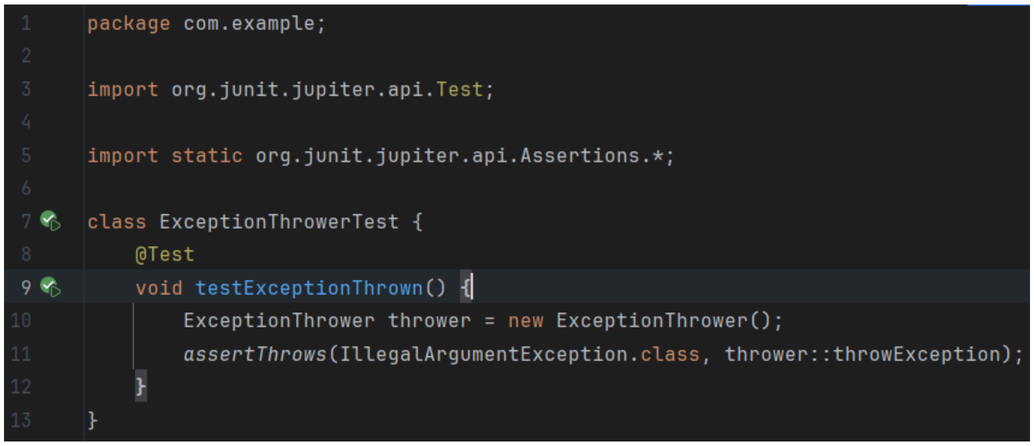
1. Create a class `ExceptionThrower` with a method `throwException`.

2. Write a test class `ExceptionThrowerTest` that tests the method for the expected exception.



```java
package com.example;

2 usages
public class ExceptionThrower {
    1 usage
    public void throwException() throws IllegalArgumentException{
        throw new IllegalArgumentException("This is an exception");
    }
}
```

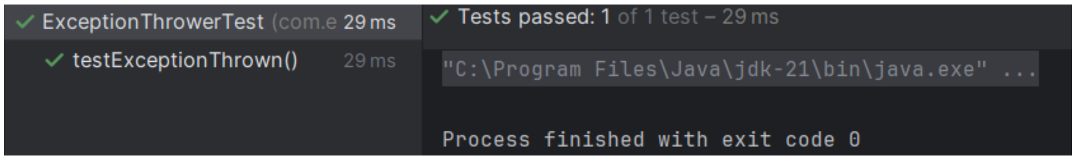


```java
package com.example;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class ExceptionThrowerTest {
    @Test
    void testExceptionThrown() {
        ExceptionThrower thrower = new ExceptionThrower();
        assertThrows(IllegalArgumentException.class, thrower::throwException);
    }
}
```



```
✓ ExceptionThrowerTest (com.e 29 ms    ✓ Tests passed: 1 of 1 test – 29 ms
  ✓ testExceptionThrown()      29 ms    "C:\Program Files\Java\jdk-21\bin\java.exe" ...

                                         Process finished with exit code 0
```

Exercise 5: Timeout and Performance Testing

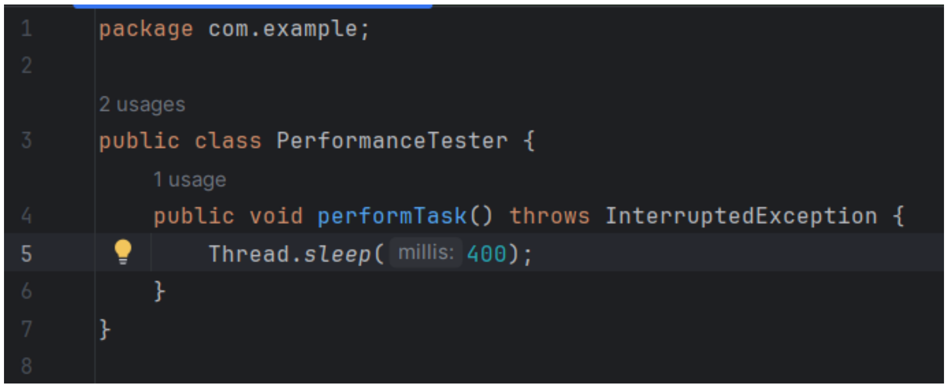
Scenario:

You want to ensure that a method completes within a specified time limit.

Steps:

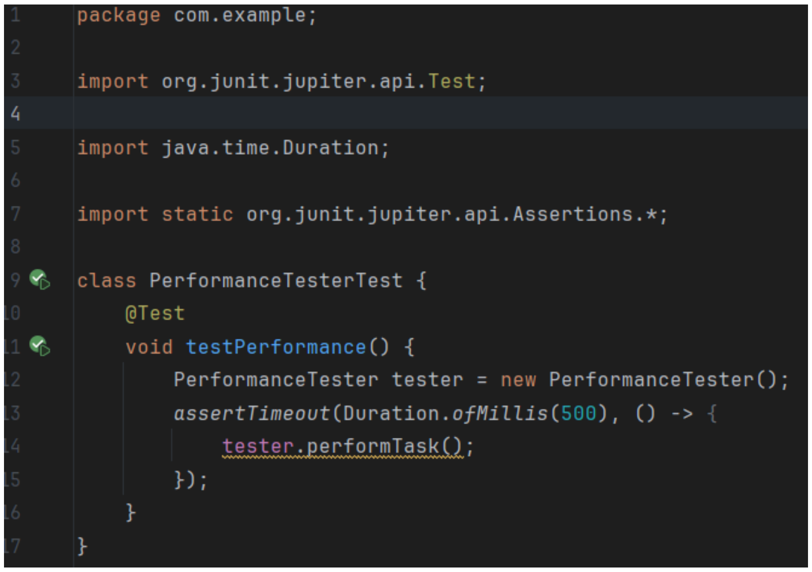
1. Create a class `PerformanceTester` with a method `performTask`.

2. Write a test class `PerformanceTesterTest` that tests the method for timeout.



```java
package com.example;

2 usages
public class PerformanceTester {
    1 usage
    public void performTask() throws InterruptedException {
        Thread.sleep( millis: 400);
    }
}
```



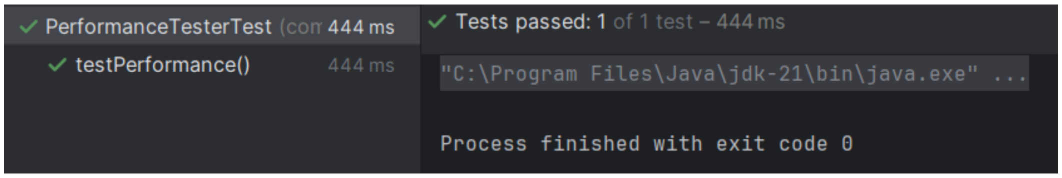
```java
package com.example;

import org.junit.jupiter.api.Test;

import java.time.Duration;

import static org.junit.jupiter.api.Assertions.*;

class PerformanceTesterTest {
    @Test
    void testPerformance() {
        PerformanceTester tester = new PerformanceTester();
        assertTimeout(Duration.ofMillis(500), () -> {
            tester.performTask();
        });
    }
}
```



```
✓ PerformanceTesterTest (com 444 ms    ✓ Tests passed: 1 of 1 test – 444 ms
  ✓ testPerformance()         444 ms    "C:\Program Files\Java\jdk-21\bin\java.exe" ...

                                         Process finished with exit code 0
```